Project 3

June 1, 2021

1 CS M148 Final Project

1.1 Introduction

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. The total cost of stroke in the US was 103.5 billion dollars according to 2016 US dollar values. 68.5 billion dollars or 66% of total cost was accounted for by indirect cost from underemployment and premature death. Age groups 45-64 years accounted for the greatest stroke related direct cost.

1.2 Challenge

This project is about being able to predict whether a patient is likely to get a stroke based on the input parameters available to us. Use features like gender, age, various medical conditions, and smoking status to build a model that helps you decide if a person is likely to experience a stroke event in the near future.

You will serve as data scientists hired by the UCLA hospital. You will be asked to develop a predictive model for this task and report out your findings to them. This project will include both a structured component, where much like Projects 1 and 2, you will be given a specific set of instructions to complete.

1.3 Attribute Information

1. id: unique identifier
2. gender: "Male", "Female" or "Other"
3. age: age of the patient
4. hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5. heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6. ever_married: "No" or "Yes"
7. work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8. Residence_type: "Rural" or "Urban"
9. avg_glucose_level: average glucose level in blood
10. bmi: body mass index
11. smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12. stroke: 1 if the patient had a stroke or 0 if not

1.4 Part 0: Setting Up

[20]:
```
#Here are a set of libraries we imported to complete this assignment.
#Feel free to use these or equivalent libraries for your implementation
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph
import os
import seaborn as sns # used for plot interactive graph.
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import sklearn.metrics.cluster as smc
from sklearn import model_selection
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier

from matplotlib import pyplot
import itertools

%matplotlib inline

import random

random.seed(42)
```

[21]:
```
# Helper function allowing you to export a graph
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

[22]:
```
# Helper function that allows you to draw nicely formatted confusion matrices
def draw_confusion_matrix(y, yhat, classes, plot_name):
    '''
        Draws a confusion matrix for the given target and predictions
        Adapted from scikit-learn and discussion example.
    '''
    plt.cla()
```

```
    plt.clf()
    matrix = confusion_matrix(y, yhat)
    plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.colorbar()
    num_classes = len(classes)
    plt.xticks(np.arange(num_classes), classes, rotation=90)
    plt.yticks(np.arange(num_classes), classes)

    fmt = 'd'
    thresh = matrix.max() / 2.
    for i, j in itertools.product(range(matrix.shape[0]), range(matrix.
↪shape[1])):
        plt.text(j, i, format(matrix[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if matrix[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    save_fig(plot_name)
    plt.show()
```

[23]:
```
data = pd.read_csv("healthcare-dataset-stroke-data.csv")
```

[24]:
```
data.head()
```

[24]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

[25]:
```
data.describe()
```

[25]:
```
                 id          age  hypertension  heart_disease  \
count   5110.000000  5110.000000   5110.000000    5110.000000
mean   36517.829354    43.226614      0.097456       0.054012
std    21161.721625    22.612647      0.296607       0.226063
min       67.000000     0.080000      0.000000       0.000000
25%    17741.250000    25.000000      0.000000       0.000000
50%    36932.000000    45.000000      0.000000       0.000000
75%    54682.000000    61.000000      0.000000       0.000000
max    72940.000000    82.000000      1.000000       1.000000

       avg_glucose_level          bmi       stroke
count        5110.000000  4909.000000  5110.000000
mean          106.147677    28.893237     0.048728
std            45.283560     7.854067     0.215320
min            55.120000    10.300000     0.000000
25%            77.245000    23.500000     0.000000
50%            91.885000    28.100000     0.000000
75%           114.090000    33.100000     0.000000
max           271.740000    97.600000     1.000000
```

[26]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

1.5 Part 1: Basic Statistics

[27]:
```
# Use label encoder for categorical variables with just 2 possible values.
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data["stroke"] = le.fit_transform(data["stroke"])
data["ever_married"] = le.fit_transform(data["ever_married"])
data["Residence_type"] = le.fit_transform(data["Residence_type"])

# We also don't need the id column.
data.drop(['id'], axis=1, inplace=True)

data.info()
```

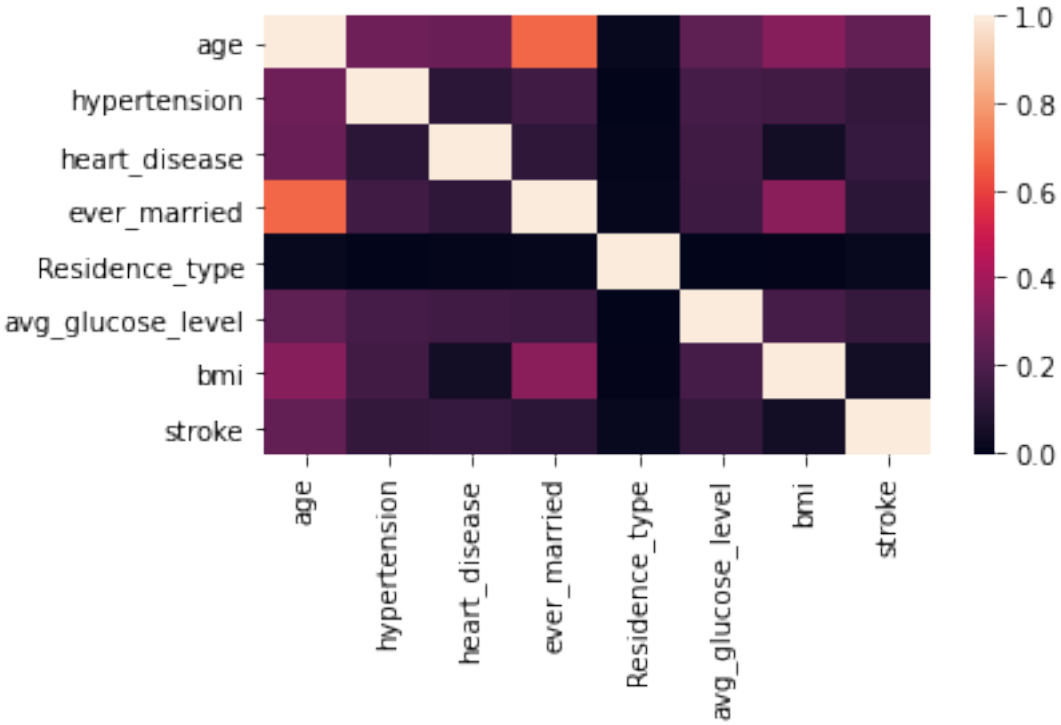
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             5110 non-null   object
 1   age                5110 non-null   float64
 2   hypertension       5110 non-null   int64
 3   heart_disease      5110 non-null   int64
 4   ever_married       5110 non-null   int32
 5   work_type          5110 non-null   object
 6   Residence_type     5110 non-null   int32
 7   avg_glucose_level  5110 non-null   float64
 8   bmi                4909 non-null   float64
 9   smoking_status     5110 non-null   object
 10  stroke             5110 non-null   int64
dtypes: float64(3), int32(2), int64(3), object(3)
memory usage: 399.3+ KB
```

[28]:
```
corr_matrix = data.corr()
sns.heatmap(corr_matrix, xticklabels=corr_matrix.columns,␣
 ↪yticklabels=corr_matrix.columns)
save_fig("correlation_plot")
corr_matrix["stroke"]
```

```
Saving figure correlation_plot
```

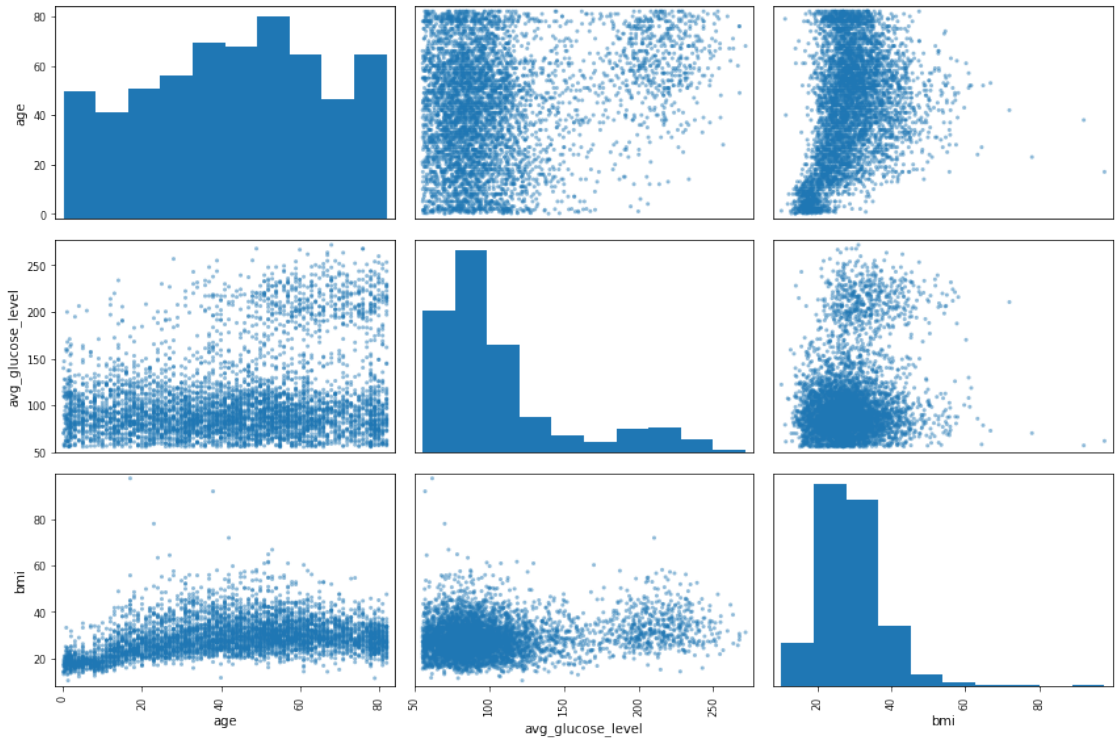
[28]:
```
age                  0.245257
hypertension         0.127904
heart_disease        0.134914
ever_married         0.108340
Residence_type       0.015458
```

```
avg_glucose_level    0.131945
bmi                  0.042374
stroke               1.000000
Name: stroke, dtype: float64
```



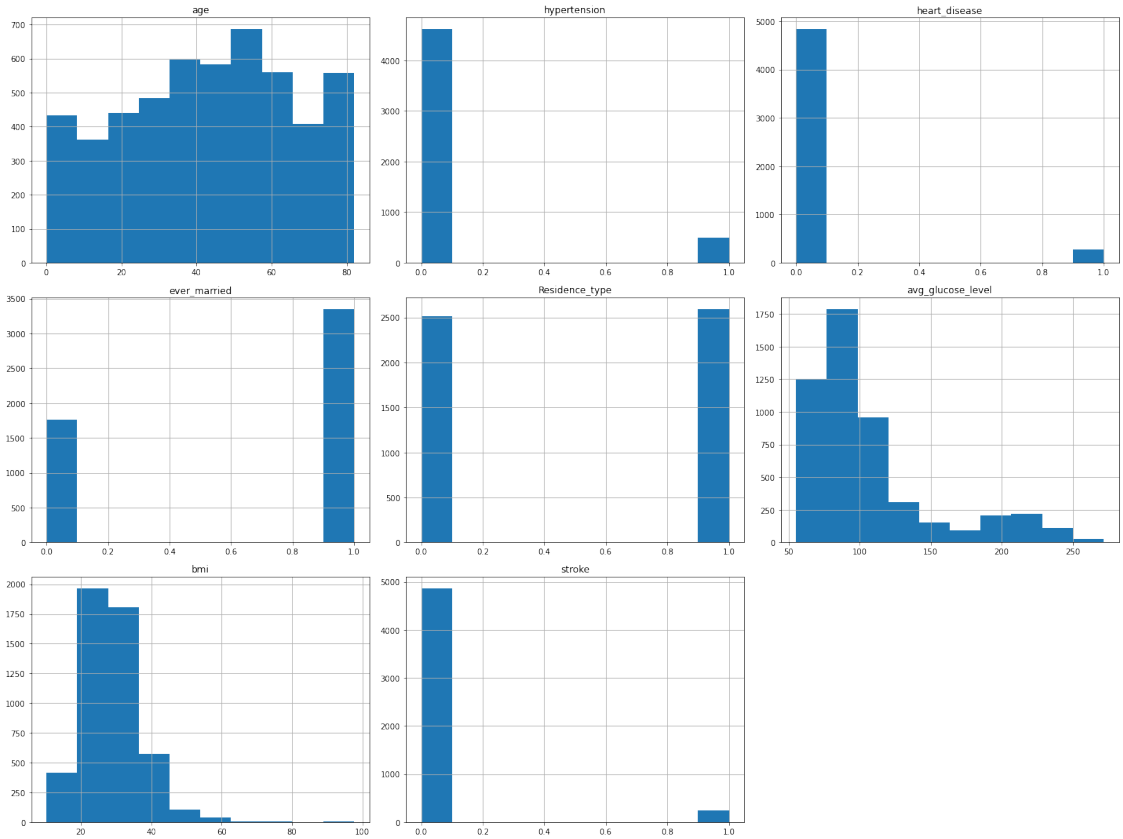
[29]:
```
from pandas.plotting import scatter_matrix
attributes = ["age", "avg_glucose_level", "bmi"]
scatter_matrix(data[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```

```
Saving figure scatter_matrix_plot
```



[30]:
```
data.hist(bins=10, figsize=(20,15))
save_fig("feature_histograms")
```

```
Saving figure feature_histograms
```

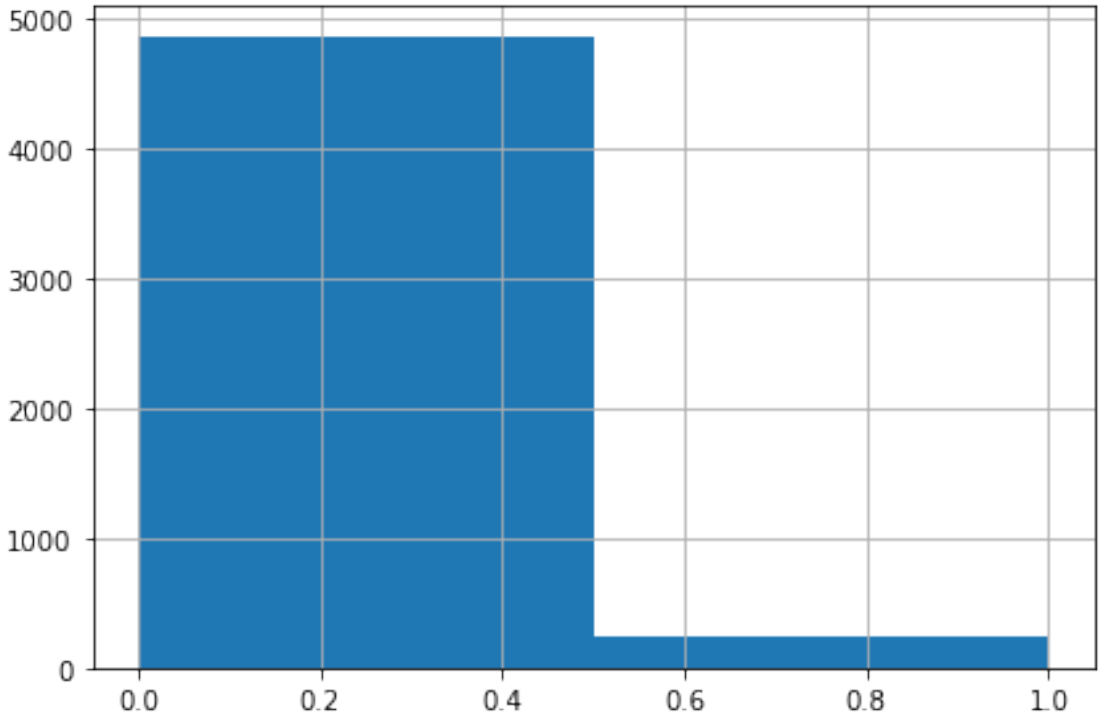


From the above, we see that bmi, age, and avg_glucose_level are gradients while hypertension, heart_disease, ever_married, Residence_type, and stroke are binary. Since stroke is our response variable, we know that we are dealing with a classification problem here.

[31]:
```
# Histogram of Healthy (0) vs. Sick (1)
data['stroke'].hist(bins=2)

num_stroke = 0
for i in data['stroke']:
    if i == 1:
        num_stroke += 1
print("number of individuals with stroke:", num_stroke)
print("number of individuals without stroke:", data['stroke'].size - num_stroke)
save_fig("stroke_histogram")
```

```
number of individuals with stroke: 249
number of individuals without stroke: 4861
Saving figure stroke_histogram
```



[32]:
```
from sklearn.utils import resample

df_majority = data[data.stroke == 0]
df_minority = data[data.stroke == 1]

df_majority_downsampled = resample(df_majority, replace=False,␣
 ↪n_samples=num_stroke, random_state=42)
df_downsampled = pd.concat([df_majority_downsampled, df_minority])

df_downsampled.stroke.value_counts()
```

[32]:
```
1    249
0    249
Name: stroke, dtype: int64
```

[33]:
```
# Our target variable is stroke (1 if the patient had a stroke and 0 otherwise).
y = df_downsampled["stroke"]

stroke = df_downsampled.copy(deep=True)
stroke.drop(["stroke"], axis=1, inplace=True)
stroke.head()
```

[33]:
```
      gender   age  hypertension  heart_disease  ever_married      work_type  \
2435    Male  44.0             0              0             1        Private
3600  Female  75.0             1              0             1  Self-employed
2900  Female  56.0             0              0             1        Private
2353  Female   5.0             0              0             0       children
4060  Female  69.0             0              0             1  Self-employed

      Residence_type  avg_glucose_level   bmi   smoking_status
2435               1              80.75  30.9     never smoked
3600               0             219.82  29.5  formerly smoked
2900               1              94.19  25.7     never smoked
2353               1             122.25  16.7          Unknown
4060               0             110.96  25.9     never smoked
```

1.6 Part 2: Data Feature Extraction

[34]:
```
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin

numerical_features = ["age", "hypertension", "heart_disease", "ever_married",␣
 ↪"Residence_type", "avg_glucose_level", "bmi"]
categorical_features = ["gender", "work_type", "smoking_status"]
age_idx, bmi_idx, glucose_idx = 0, 6, 5

class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    stroke["age_over_bmi"] = stroke["age"] / stroke["bmi"]
    stroke["age_over_glucose"] = stroke["age"] / stroke["avg_glucose_level"]
    stroke["bmi_over_glucose"] = stroke["bmi"] / stroke["avg_glucose_level"]
    '''
    def __init__(self, add_augmented_features = True):
        self.add_augmented_features = add_augmented_features

    def fit(self, X, y=None):
        return self  # nothing else to do

    def transform(self, X):
        age_over_bmi = X[:, age_idx] / X[:, bmi_idx]
        age_over_glucose = X[:, age_idx] / X[:, glucose_idx]
        bmi_over_glucose = X[:, bmi_idx] / X[:, glucose_idx]
```

```
        if self.add_augmented_features:
            return np.c_[X, age_over_bmi, age_over_glucose, bmi_over_glucose]
        else:
            return np.c_[X]

# this will be are numirical pipeline
# 1. impute, 2. augment the feature set 3. normalize using StandardScaler()
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

stroke_prepared = full_pipeline.fit_transform(stroke)
stroke_prepared
```

[34]:
```
array([[-0.5414508 , -0.46966822, -0.37011661, …,  0.        ,
         1.        ,  0.        ],
       [ 0.91248638,  2.12916259, -0.37011661, …,  1.        ,
         0.        ,  0.        ],
       [ 0.02136359, -0.46966822, -0.37011661, …,  0.        ,
         1.        ,  0.        ],
       …,
       [ 0.91248638, -0.46966822, -0.37011661, …,  1.        ,
         0.        ,  0.        ],
       [ 0.72488158,  2.12916259, -0.37011661, …,  0.        ,
         0.        ,  0.        ],
       [ 1.05318998, -0.46966822, -0.37011661, …,  0.        ,
         0.        ,  0.        ]])
```

1.7 Part 3: Logistic Regression

Before we move on, we must split our pipelined data into training set and testing set. Let's try an 80/20 split and print out each set's dimensions to make sure the data was split properly.

[35]:
```
X_train, X_test, y_train, y_test = train_test_split(stroke_prepared, y,␣
 ↪test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

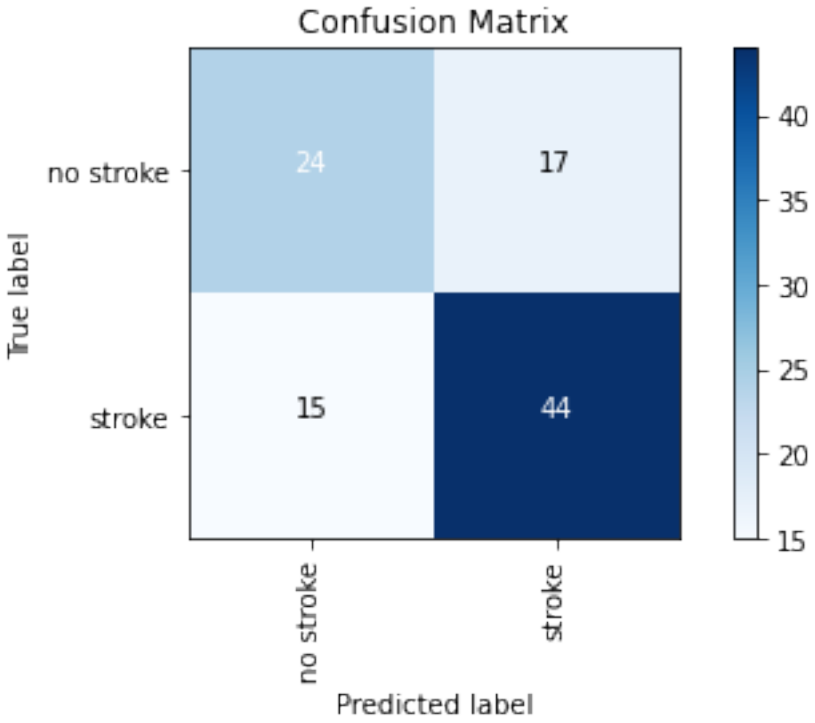
```
(398, 21) (398,)
(100, 21) (100,)
```

The dimensions match, so now we can move on and perform logistic regression!

[36]:
```
log_reg = LogisticRegression(solver='liblinear')
log_reg.fit(X_train, y_train)
predicted = log_reg.predict(X_test)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(y_test, predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(y_test, predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(y_test, predicted)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(y_test, predicted)))
print("\nConfusion Matrix:")
draw_confusion_matrix(y_test, predicted, ["no stroke", "stroke"],␣
↪"logistic_regression")
```

```
Accuracy:    0.680000
Precision:   0.721311
Recall:      0.745763
F1 Score:    0.733333

Confusion Matrix:
Saving figure logistic_regression
```



1.8 Part 4: Principal Component Analysis

[38]:
```
from sklearn import decomposition

pca = decomposition.PCA(n_components=6)
stroke_pca = pca.fit_transform(stroke_prepared)
print(stroke_pca.shape)
stroke_pca
```

```
(498, 6)
```

[38]:
```
array([[ 0.6079601 ,  0.99067498,  0.31626082, -0.96023587, -0.33244747,
        -0.327078  ],
       [-1.44459372, -2.11776559,  0.73550275,  1.80465949,  0.55403451,
        -1.27913389],
       [-0.1832898 ,  0.32031429, -0.74170939, -0.68951798, -0.58875659,
        -0.61449717],
       …,
       [-1.58521823,  1.45523429, -0.53101436,  0.71805408, -0.79474391,
         0.39927266],
       [-1.56306343,  0.68778252, -0.06832689,  1.70217153,  1.1557491 ,
        -0.64714356],
       [-2.20312003,  0.59411349, -2.34259369,  1.15352128, -1.04901341,
         0.30713573]])
```

[39]:
```
new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(stroke_pca,␣
↪y, test_size=0.2)
print(new_X_train.shape, new_y_train.shape)
print(new_X_test.shape, new_y_test.shape)
```

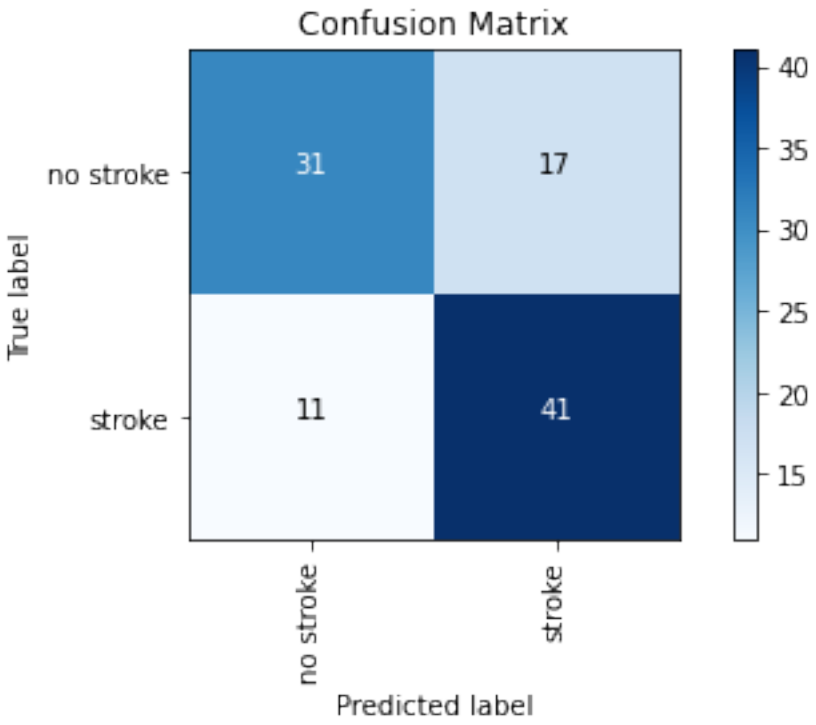
```
(398, 6) (398,)
(100, 6) (100,)
```

[41]:
```
log_reg = LogisticRegression(solver='liblinear')
log_reg.fit(new_X_train, new_y_train)
predicted = log_reg.predict(new_X_test)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(new_y_test, predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(new_y_test,␣
↪predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(new_y_test, predicted)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(new_y_test, predicted)))
print("\nConfusion Matrix:")
draw_confusion_matrix(new_y_test, predicted, ["no stroke", "stroke"],␣
↪"pca_logistic_regression")
```

```
Accuracy:    0.720000
Precision:   0.706897
Recall:      0.788462
```

```
F1 Score:    0.745455

Confusion Matrix:
Saving figure pca_logistic_regression
```



1.9 Part 5: Ensemble Method

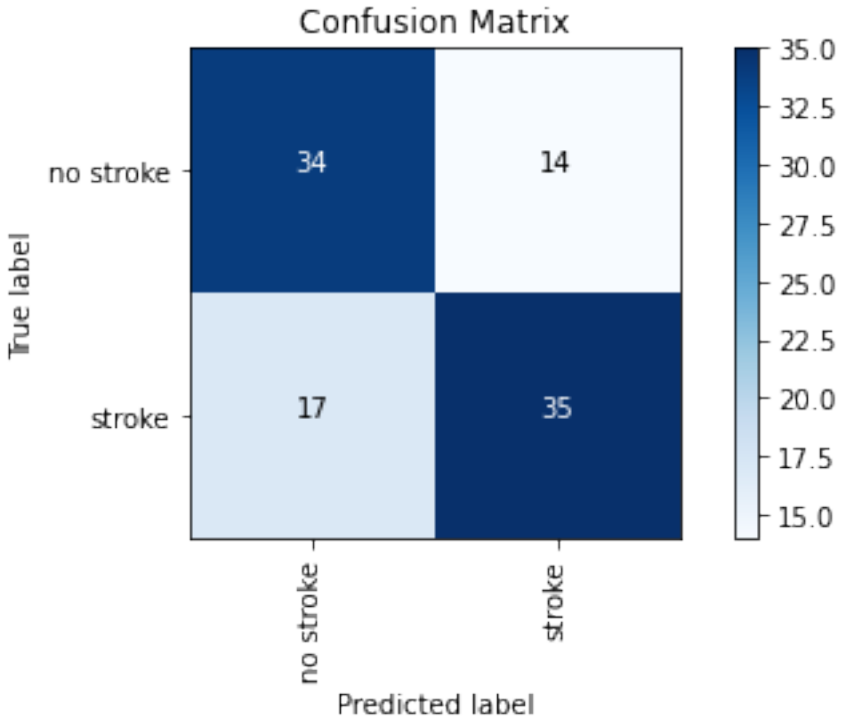
Before we jump into implementing an ensemble method, let's see what a decision tree looks like on our data.

[47]:
```
dt = DecisionTreeClassifier()
dt.fit(new_X_train, new_y_train)
print("dt.score(new_X_train, new_y_train):", dt.score(new_X_train, new_y_train))
print("dt.score(new_X_test, new_y_test):", dt.score(new_X_test, new_y_test))

predicted = dt.predict(new_X_test)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(new_y_test, predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(new_y_test,␣
 ↪predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(new_y_test, predicted)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(new_y_test, predicted)))
print("\nConfusion Matrix:")
draw_confusion_matrix(new_y_test, predicted, ["no stroke", "stroke"],␣
 ↪"single_decision_tree")
```

```
dt.score(new_X_train, new_y_train): 1.0
dt.score(new_X_test, new_y_test): 0.69
Accuracy:    0.690000
Precision:   0.714286
Recall:      0.673077
F1 Score:    0.693069

Confusion Matrix:
Saving figure single_decision_tree
```



As we can see above, a single decision tree overfits our training data, and it performs poorly on our testing data with only 62% accuracy. Let's see if we can do better by trying out different ensemble methods. The first one we'll try is Random Forest, which is an ensemble of decision trees.

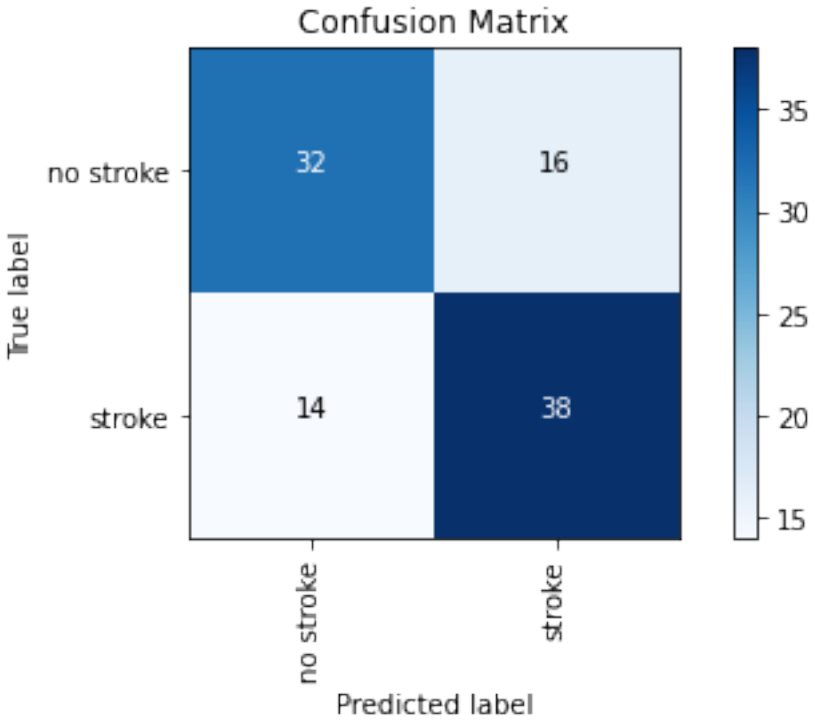
[49]:
```
rf = RandomForestClassifier(n_estimators=10) # Let's try 10 decision trees.
rf.fit(new_X_train, new_y_train)
print("rf.score(new_X_train, new_y_train):", rf.score(new_X_train, new_y_train))
print("rf.score(new_X_test, new_y_test):", rf.score(new_X_test, new_y_test))

predicted = rf.predict(new_X_test)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(new_y_test, predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(new_y_test,␣
 ↪predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(new_y_test, predicted)))
```

```
print("%-12s %f" % ('F1 Score:', metrics.f1_score(new_y_test, predicted)))
print("\nConfusion Matrix:")
draw_confusion_matrix(new_y_test, predicted, ["no stroke", "stroke"],␣
↪"random_forest")
```

```
rf.score(new_X_train, new_y_train): 0.9899497487437185
rf.score(new_X_test, new_y_test): 0.7
Accuracy:    0.700000
Precision:   0.703704
Recall:      0.730769
F1 Score:    0.716981

Confusion Matrix:
Saving figure random_forest
```



By using a random forest with 10 decision trees, we are able to reduce overfitting and improve our accuracy on the test set by about 0.1. Now let's move on to bagging, which uses multiple models of the same learning algorithm trained with subsets of data randomly picked from the training dataset.

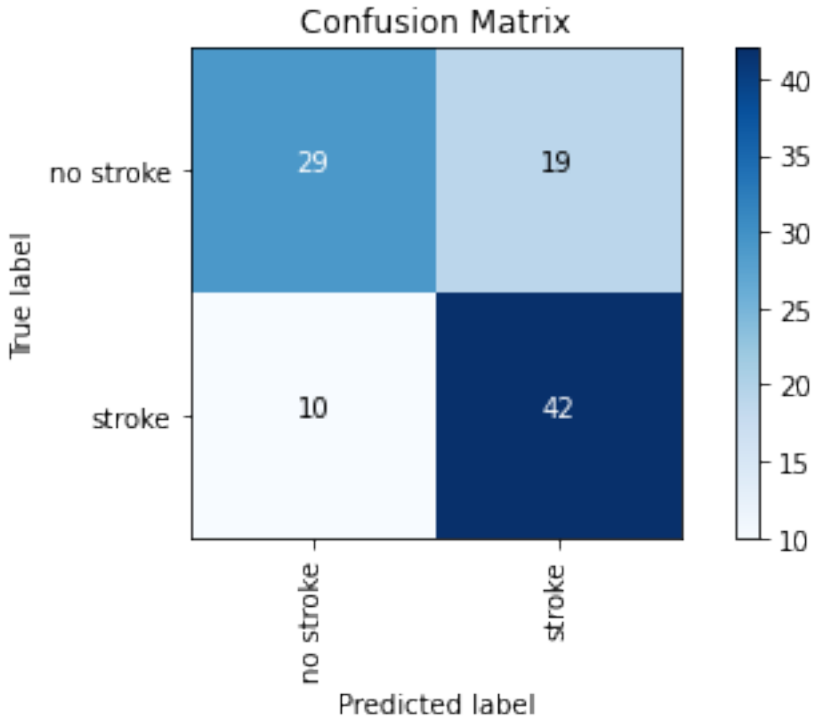
[50]:
```
bg = BaggingClassifier(DecisionTreeClassifier(), max_samples=0.3,␣
↪n_estimators=20)
bg.fit(new_X_train, new_y_train)
print("bg.score(new_X_train, new_y_train):", bg.score(new_X_train, new_y_train))
```

```
print("bg.score(new_X_test, new_y_test):", bg.score(new_X_test, new_y_test))

predicted = bg.predict(new_X_test)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(new_y_test, predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(new_y_test,␣
 ↪predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(new_y_test, predicted)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(new_y_test, predicted)))
print("\nConfusion Matrix:")
draw_confusion_matrix(new_y_test, predicted, ["no stroke", "stroke"], "bagging")
```

```
bg.score(new_X_train, new_y_train): 0.8592964824120602
bg.score(new_X_test, new_y_test): 0.71
Accuracy:    0.710000
Precision:   0.688525
Recall:      0.807692
F1 Score:    0.743363

Confusion Matrix:
Saving figure bagging
```



Through bagging, we were able to limit overfitting even more and achieve a higher accuracy score than random forest. We used 20 decision trees, and each bag could contain a maximum of 30% of the training dataset.

1.10 Part 6: Neural Net

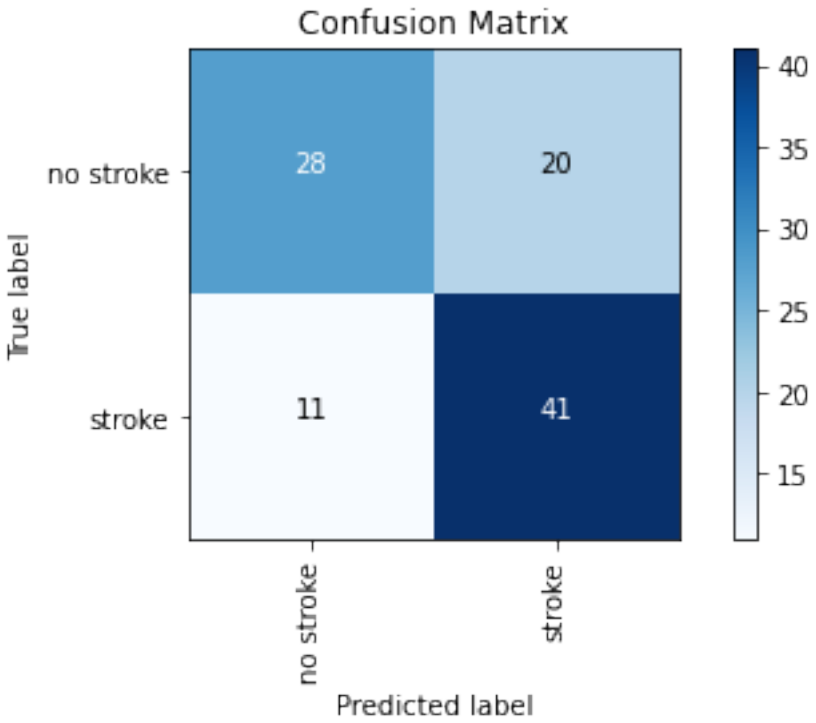
[51]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(max_iter=2400, activation='relu')
mlp.fit(new_X_train, new_y_train)

predicted = mlp.predict(new_X_test)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(new_y_test, predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(new_y_test,␣
 ↪predicted)))
print("%-12s %f" % ('Recall:', metrics.recall_score(new_y_test, predicted)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(new_y_test, predicted)))
print("\nConfusion Matrix:")
draw_confusion_matrix(new_y_test, predicted, ["no stroke", "stroke"],␣
 ↪"neural_net")
```

```
Accuracy:    0.690000
Precision:   0.672131
Recall:      0.788462
F1 Score:    0.725664

Confusion Matrix:
Saving figure neural_net
```



1.11 Part 7: K-Fold Cross Validation

The logistic regression using PCA, the ensemble method using bagging, and the neural net using the MLP Classifier were the three models that performed best out of all the models I tried for this project. I decided to run K-Fold cross validation on all three of them to determine which one performed best on our dataset. Let's start with logistic regression using PCA:

[52]:
```
kfold = model_selection.KFold(n_splits=16, random_state=42, shuffle=True)
model_kfold = LogisticRegression(solver='liblinear')
results_kfold = model_selection.cross_val_score(model_kfold, stroke_pca,␣
↪df_downsampled["stroke"], cv=kfold)

print("For logistic regression using PCA, our mean accuracy across folds is %.
↪2f%%" % (results_kfold.mean()*100.0))
```

```
For logistic regression using PCA, our mean accuracy across folds is 75.53%
```

Next, let's run K-Fold cross validation on the ensemble method using bagging.

[53]:
```
kfold = model_selection.KFold(n_splits=16, random_state=42, shuffle=True)
model_kfold = BaggingClassifier(DecisionTreeClassifier(), max_samples=0.3,␣
↪n_estimators=20)
results_kfold = model_selection.cross_val_score(model_kfold, stroke_pca,␣
↪df_downsampled["stroke"], cv=kfold)

print("For ensemble method using bagging, our mean accuracy across folds is %.
↪2f%%" % (results_kfold.mean()*100.0))
```

```
For ensemble method using bagging, our mean accuracy across folds is 74.50%
```

Finally, on the neural net using MLP Classifier:

[54]:
```
kfold = model_selection.KFold(n_splits=16, random_state=42, shuffle=True)
model_kfold = MLPClassifier(max_iter=2400, activation='relu')
results_kfold = model_selection.cross_val_score(model_kfold, stroke_pca,␣
↪df_downsampled["stroke"], cv=kfold)

print("For neural net using MLP Classifier, our mean accuracy across folds is %.
↪2f%%" % (results_kfold.mean()*100.0))
```

```
For neural net using MLP Classifier, our mean accuracy across folds is 71.28%
```

CS M148 Project 3 Report

Bonnie Liu
005300989
Discussion 1B
6/1/2021

Executive Summary

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. The total cost of stroke in the US was $103.5 billion according to 2016 US dollar values. $68.5 billion or 66% of total cost was accounted for by indirect cost from underemployment and premature death. Age groups 45-64 years accounted for the greatest stroke related direct cost.

In this project, I worked with the Stroke Prediction Dataset found on Kaggle, which contained information of 5,110 individuals pertaining whether or not they had a stroke, gender, age, bmi, etc. My goal in this project is to leverage the data provided to come up with a prediction model that can accurately predict if an individual is at risk for stroke.

During the process of exploratory data analysis, I found that I was dealing with an imbalanced classification problem since only about 5% of the individuals in the dataset had a stroke. To deal with this, I downsampled the majority class. Additionally, the bmi feature was the only column with null values, so I imputed them with the median bmi value.

For feature augmentation, I explored how age, bmi, and average glucose level were all related to one another. For the binary categorical features like ever_married or Residence_type, I replaced the options with 0 and 1. For the categorical features with more than 2 categories like gender, work_type and smoking_status, I decided to one-hot encode them since the dimensionality of our dataset would not explode even after doing so. I ended up with 22 dimensions.

I then proceeded to try out a couple of different models and compared their performances. Although I reported the accuracy, precision, recall, F1 score, and confusion matrix for each model, I decided that the best metric for comparing model performance is F1 score since it takes into account both recall and precision.

The first method I tried was logistic regression on my augmented dataset. Then I used principal component analysis to reduce the dimensionality of my data from 22 features to 6 features. Doing so actually improved the performance of my logistic regression model.

Next, I tried two different ensemble methods using Decision Trees: Random Forest and bagging. Both performed better than a singular decision tree since a model based on a single decision tree tends to overfit the model. By taking the average over multiple decision trees or splitting the testing data into the different “bags”, we are able to prevent overfitting and improve our model performance.

Then I tried neural networks using sklearn's MLP Classifier. Initially, I kept getting the Convergence Warning, which means that optimization wasn't reached under the maximum number of iterations set. I continued increasing the parameter max_iter until it hit 2400 and consistently ran to completion. I also entertained different activation methods like sigmoid and tanh but ultimately stuck with ReLU because according to the sklearn documentation, ReLU works well for smaller datasets like ours. I achieved mediocre results with the MLP Classifier, but I wanted to see which model truly excelled by using K-fold cross validation.

I used 16 folds for my cross validation and did not want to increase the number of folds even more since the neural net took a significant amount of time to run to completion. Upon reporting the mean accuracy across folds, I found that logistic regression using PCA had the highest mean accuracy across folds with the ensemble method using bagging coming in close second.

Introduction

Background

According to the Center for Disease Control, stroke is a leading cause of death and a major cause of disability in the United States. Every year, about 795,000 people suffer from a stroke. 1 in every 6 deaths from cardiovascular disease was due to stroke in 2018.

Risk

If an individual has suffered from a stroke before, they are more likely to get it again. Some other conditions that increase the risk for stroke include high blood pressure, high cholesterol, heart disease, diabetes, sickle cell disease. There are also behaviors that increase the risk for stroke, including unhealthy diet, physical inactivity, obesity, too much alcohol, and tobacco usage. Other characteristics that increase the risk for stroke include whether someone else in their family has had a stroke before, age, sex, and race or ethnicity.

Attribute Information

1. id: unique identifier
2. gender: "Male", "Female" or "Other"
3. age: age of the patient
4. hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5. heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6. ever_married: "No" or "Yes"
7. work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8. Residence_type: "Rural" or "Urban"
9. avg_glucose_level: average glucose level in blood
10. bmi: body mass index
11. smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12. stroke: 1 if the patient had a stroke or 0 if not

Methodology

Part 0: Setting Up

I started by importing a bunch of Python modules and defining some functions that will come in handy later on in the process. Please note that these were the same as those provided in Project 2. Next, I read healthcare-dataset-stroke-data.csv.

Part 1: Basic Statistics

I ran the correlation matrix, and I found that ever_married and age are quite strongly correlated, which makes sense since people are more likely to be married after a certain age. Then I plotted scatter plots of the 3 numerical variables against each other, and I found that there seems to be a general positive correlation between age and bmi. By plotting some histograms, I found that our data is unbalanced. Just under 5% of the individuals in our dataset had stroke. This is problematic because in the future, our model could just guess “no stroke” 100% of the time and still get the correct answer 95% of the time. To deal with this issue, I researched a little and found that I had two options:

- Upsample the minority class.
    - Advantage: no loss of information
    - Disadvantage: possibility of overfitting
- Downsample the majority class
    - Advantage: runtime improved
    - Disadvantage: loss of information in majority class

I decided to downsample since we had 249 data points in our minority class, which I felt was enough for building a model.

Part 2: Data Feature Extraction

Upon reading healthcare-dataset-stroke-data.csv, the info() function showed that the bmi column had 201 missing values. To deal with this, I entertained a couple of options:

- Drop all rows with missing bmi values. 201/5110 is about 4% of the total data. This isn’t too bad, but let’s see if we can do better.
- Drop the bmi feature. Based on what I know about strokes and bmi, I have a gut feeling that these are correlated, so I decided against this option.
- Replace null bmi values with an arbitrary value like 0. While this is a quick and easy solution to our problem, I ultimately decided against this one as well since bmi is a numerical feature. Thus, we’ll want to perform linear regression on this feature, and replacing the null values with 0 will mess up our results.
- Replace null bmi values with values predicted by linear regression. This is perhaps the most effective solution, but given the fact that our dataset is quite large with 5110 entries, this may take a long time and may be computationally expensive.
- Replace null bmi values with the median or the mean. The describe() function showed that bmi data is skewed to the right, so I decided to replace null bmi values with the median rather than the mean.

I ultimately decided to replace the null bmi values with the median because of the aforementioned reasons. I also standardized our numerical features so that no single numerical feature will have a disproportionately large effect on the model. I one-hot encoded gender, work_type, and smoking_status because those were all categorical variables with a fairly low

number of categories, so our dimensionality did not inflate drastically. Lastly, I augmented the following features:

- age_over_bmi: As noted in Part 1, there seemed to be a general positive correlation between age and bmi.
- age_over_glucose: I thought it would be interesting to examine how average glucose levels and age affected each other.
- bmi_over_glucose: I thought it would be interesting to see if bmi and average glucose levels were related in any way.

I ended up with 22 columns total in my dataset after feature extraction.

Part 3: Logistic Regression

I decided to split my pipelined data in order to test my models. I allocated 80% of the data for the training set and 20% of the data for the testing set. For logistic regression, I decided to use the liblinear solver because it's good for smaller datasets. I also tried the default solver for logistic regression, which is lbfgs, and indeed, logistic regression using the liblinear solver performed better on our dataset. To analyze data, I printed out the following:

- Accuracy: the fraction of predictions our model got correct
- Precision: false positive rate
- Recall: false negative rate
- F1 score: a balance of precision and recall

I decided to go with F1 score as my main indicator of whether or not a model was good since it's the most generalized measure. A higher F1 score indicates better model performance.

Part 4: Principal Component Analysis

As mentioned above, my model had 22 columns, which is a lot, and by using principal component analysis, we can reduce the dimensionality in hopes of preventing overfitting. I played around with the parameter n_components (number of components) with values ranging from 3 to 10, and upon comparing F1 scores, I decided that n_components=6 produced the most optimal results.

Part 5: Ensemble Method

Before jumping into implementing an ensemble method, I wanted to see what a single decision tree looks like on our data. Then I proceeded to do a random forest with 10 decision trees. (I tried various values for the number of decision trees and settled at 10 because that was approximately where performance started to plateau.) Finally, I moved on to bagging by using the Bagging Classifier with the following parameters: Decision Tree Classifier, max_samples=0.3, and n_estimators=20. This meant that we would have 20 decision trees and that each "bag" could have a maximum of 30% of our training dataset. I came to these parameter values after a lot of experimenting (trial and error).

Part 6: Neural Net

I decided to use sklearn's MLP Classifier since it was the simplest to implement, but other (perhaps better) neural networks include pytorch and keras. I played around with the parameters of the MLP classifier. I kept getting ConvergenceWarning, which meant that the

optimization didn't converge under the current number of iterations, so I kept increasing max_iter until it hit 2000, which is pretty large compared to the default of 200. As a result of the large number of iterations my model had to go through, it took a while for my model to converge and my neural net to finish running. I did a little bit of research (see resources section below), and I found that ReLU converges faster than other activation functions like sigmoid and tanh, so I decided to stick with ReLU, which is the default.

Part 7: K-Fold Cross Validation

I ran K-fold cross validation on the logistic regression using PCA, the ensemble method using bagging, and the neural net using the MLP Classifier. These 3 were my best-performing models from this project, so I wanted to see how they would prefer over many iterations. I set shuffle to True and random_state to the same for all 3. For the number of folds, I set it to 16 since there were 498 rows in my data, so setting it to 16 would mean that each fold had about 31 data points. I was unwilling to create more folds because the neural net took a long time to run with 16 folds.

Part 8: Reporting Highest-Performing Model

Using the results from my K-fold cross validation, I printed out the mean accuracy across folds for each of the 3 models. I picked the model with the highest mean accuracy across folds as my best (i.e. highest-performing) model.

Results

Part 0: Setting Up

```
In [445]:  1 data.head()
```

Out[445]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

```
In [446]:  1 data.describe()
```

Out[446]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

```
In [447]:  1 data.info()
```

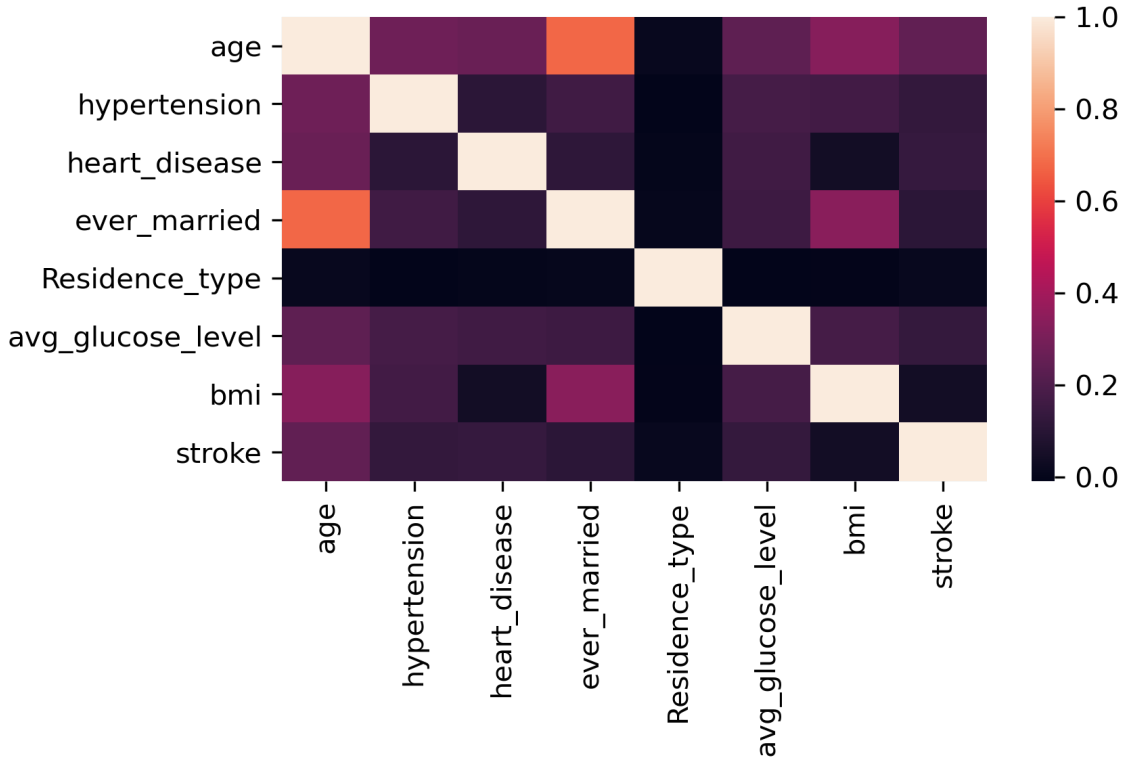
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

From the above basic Python data analysis functions, I was able to find that bmi was the only feature with null values. Additionally, I was able to conclude that age, avg_glucose_level, and bmi were the only numerical features whereas everything else was categorical.

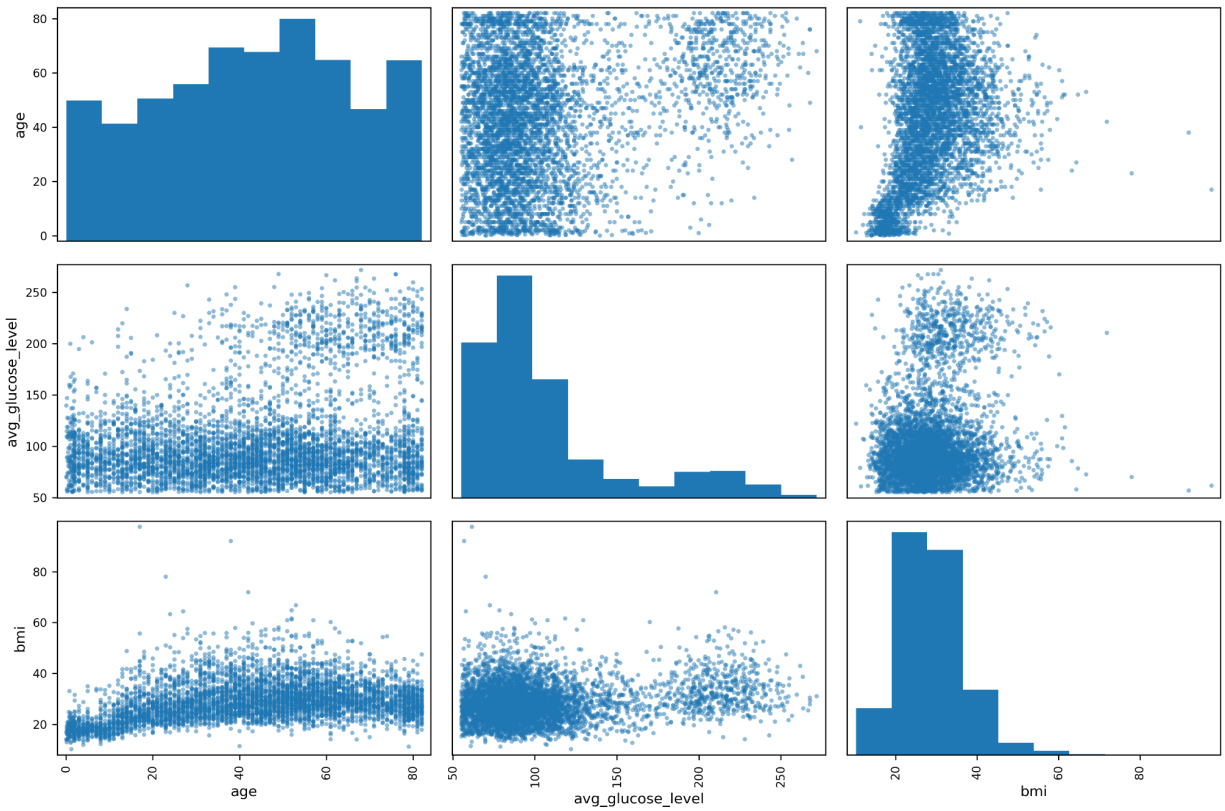
Part 1: Basic Statistics

In order to clean up the data, I used Label Encoder to change our binary categorical variables (i.e. stroke, ever_married, and Residence_type) into 0's and 1's. I also dropped the id column. This was my correlation matrix and heatmap afterwards:

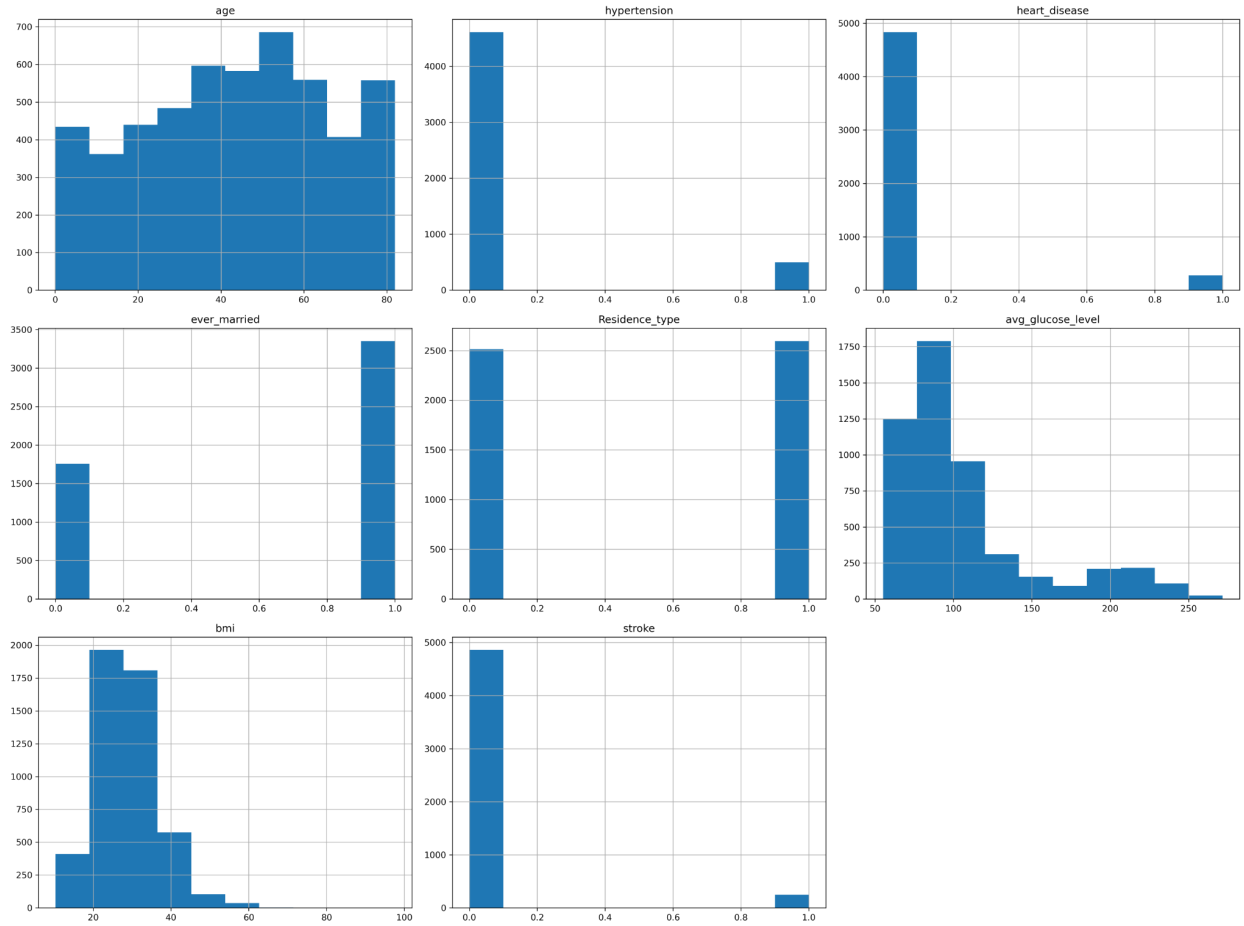
```
age                0.245257
hypertension       0.127904
heart_disease      0.134914
ever_married       0.108340
Residence_type     0.015458
avg_glucose_level  0.131945
bmi                0.042374
stroke             1.000000
Name: stroke, dtype: float64
```



As seen above, there seems to be a positive correlation between stroke and age, hypertension, heart_disease, avg_glucose_level, and marital status. I wanted to dive into the numerical features of the dataset and see if any of them were related, so I made scatterplots of age, avg_glucose_level, and bmi.



There seemed to be a positive correlation between age and bmi as well. It looks like perhaps a logarithmic relationship might exist between bmi and age. I proceeded by plotting histograms for each feature so that I could gain a better understanding of each feature individually.



As seen from above, bmi, age, and avg_glucose_level are gradients while hypertension, heart_disease, ever_married, Residence_type, and stroke are binary. Since stroke is our response variable, we know that we are dealing with a classification problem here. I decided to examine stroke in more detail, and I found that in our dataset, 249 individuals had a stroke while 4861 individuals did not. This means that just under 5% of our dataset has had a stroke and that our dataset is largely imbalanced. To deal with this, I decided to downsample the majority class as mentioned in the Methodology section. After doing so, I had 249 individuals with strokes and 249 individuals without.

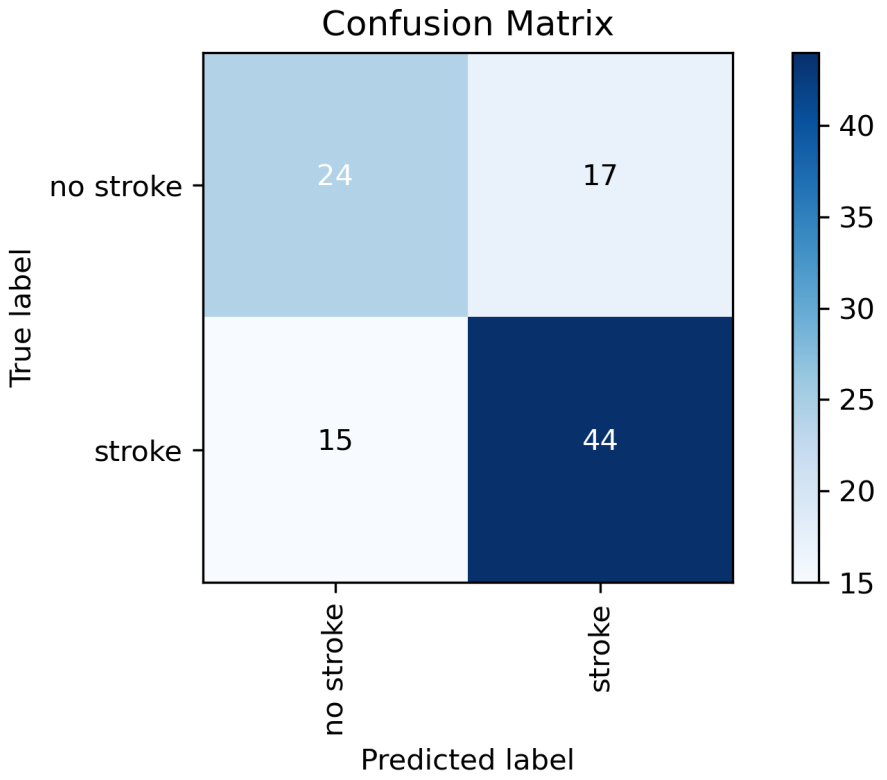
Part 2: Data Feature Extraction

After feature extraction using the pipeline, I ended up with a dataset with 498 rows and 22 columns.

Part 3: Logistic Regression

I split the dataset using an 80/20 split, so my training dataset had 398 data points while my testing dataset had 100.

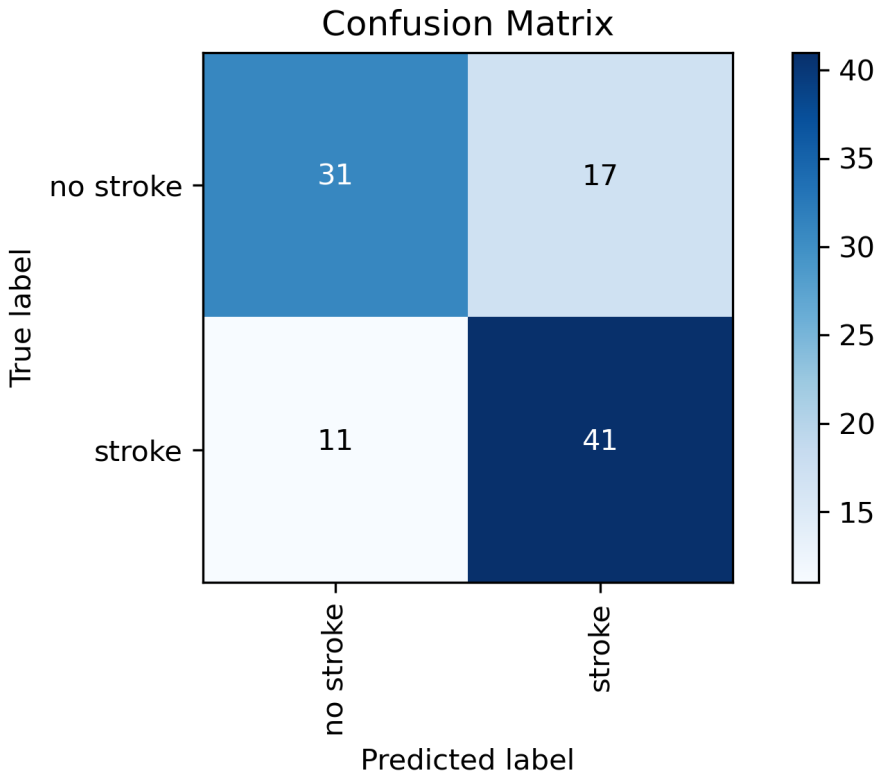
```
Accuracy:    0.680000
Precision:   0.721311
Recall:      0.745763
F1 Score:    0.733333
```



This isn't great considering the accuracy is pretty low, so let's see if we can do better after implementing Principal Component Analysis.

Part 4: Principal Component Analysis

```
Accuracy:    0.720000
Precision:   0.706897
Recall:      0.788462
F1 Score:    0.745455
```

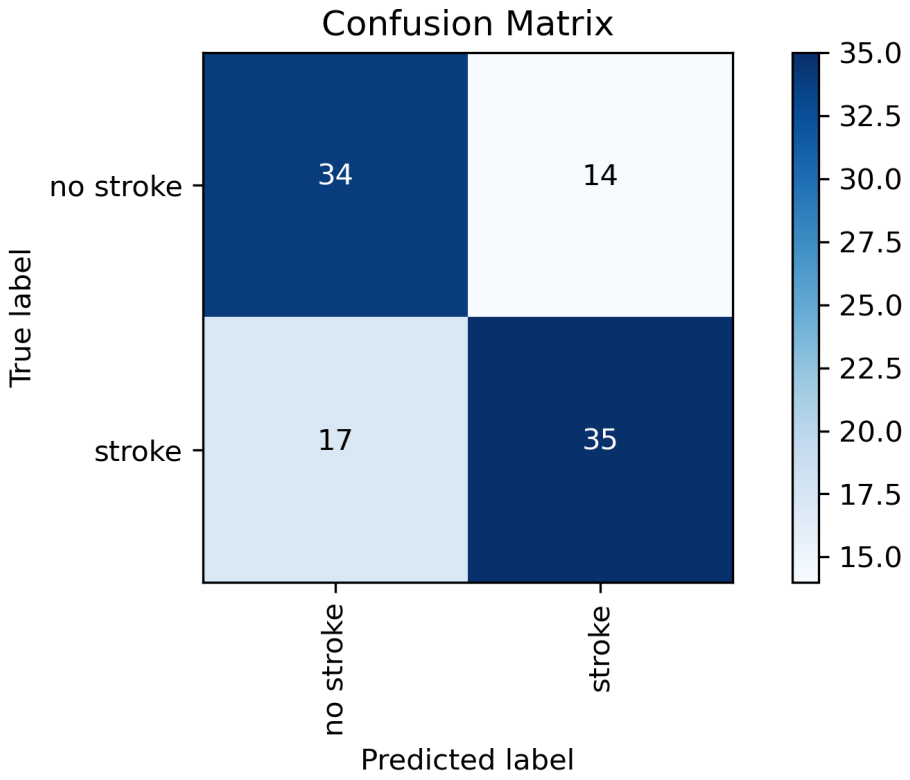


Indeed, our model improved when we reduced the dimensionality of the dataset from 22 columns to just 6. We got a better accuracy and F1 score as a result.

Part 5: Ensemble Methods

Before jumping into ensemble methods, I wanted to test out a single decision tree, and when I ran it, I got the following results:

```
Accuracy:    0.690000
Precision:   0.714286
Recall:      0.673077
F1 Score:    0.693069
```

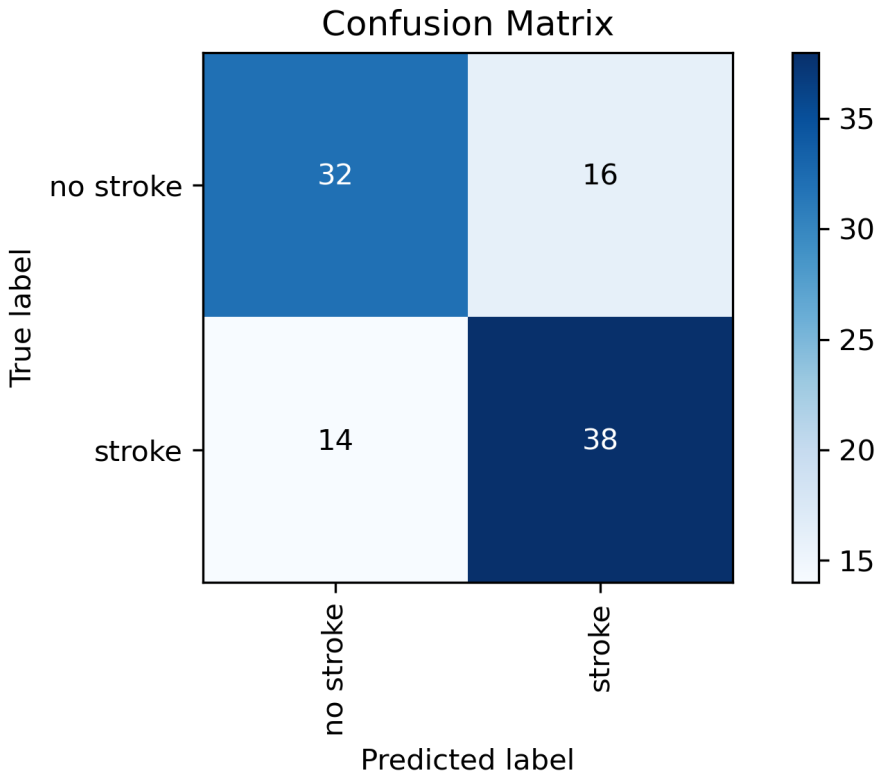


With a single decision tree, we can see that the model performance was not very good. I decided to dive into this situation a little further:

```
dt.score(new_X_train, new_y_train): 1.0
dt.score(new_X_test, new_y_test): 0.69
```

As we can see, it seems like our model is overfitting the training data. That's where ensemble methods come into play. Let's first try random forest, which is just a bunch of decision trees.

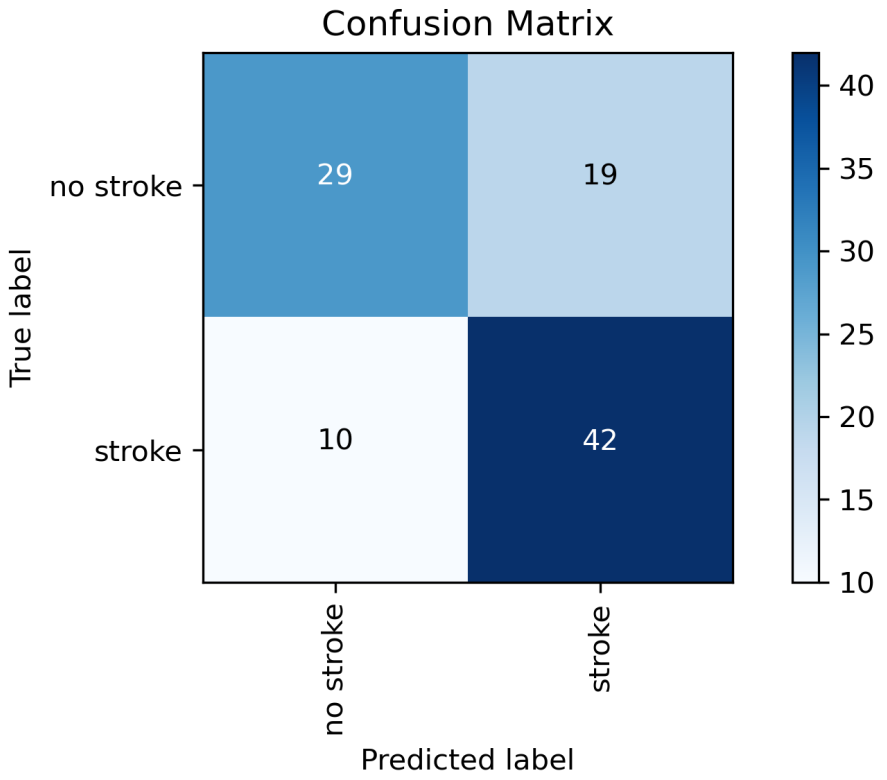
```
Accuracy:    0.700000
Precision:   0.703704
Recall:      0.730769
F1 Score:    0.716981
```




```
rf.score(new_X_train, new_y_train): 0.9899497487437185
rf.score(new_X_test, new_y_test): 0.7
```

As we can see, our random forest model does better than a single decision tree because it takes the average across all the decision trees. Our model is no longer overfitting the training data as much, which is consistent with our finding that this random forest performs better on the testing set than a single decision tree. Now let's try bagging to see if we can improve the performance even more.

```
Accuracy:    0.710000
Precision:   0.688525
Recall:      0.807692
F1 Score:    0.743363
```

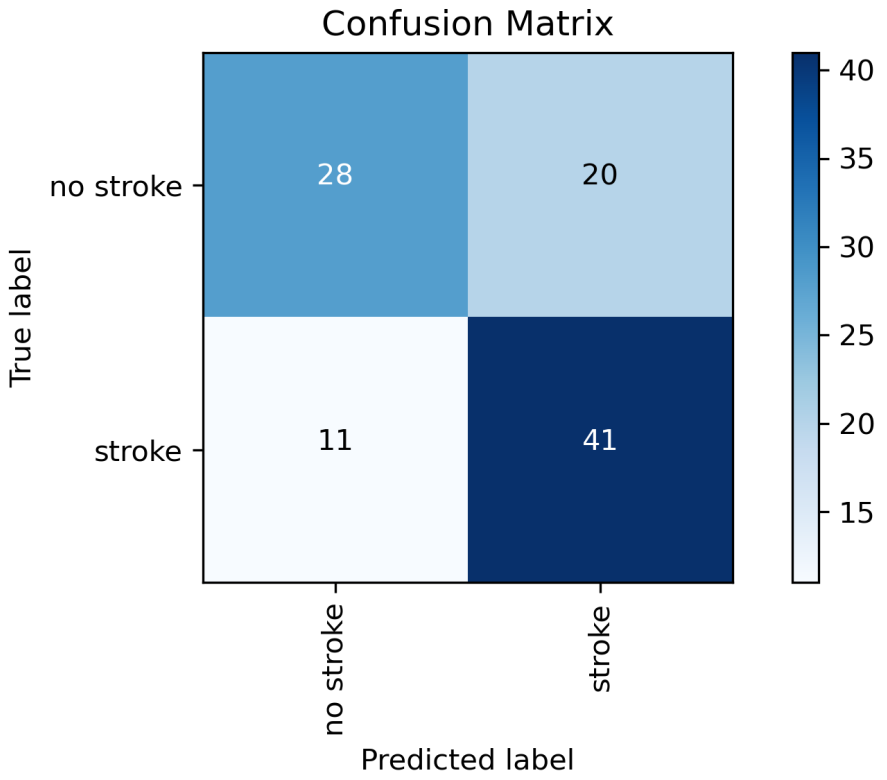



```
bg.score(new_X_train, new_y_train): 0.8592964824120602
bg.score(new_X_test, new_y_test): 0.71
```

Based on the above results, bagging drastically reduces overfitting and improves our model performance.

Part 6: Neural Net

```
Accuracy:    0.690000
Precision:   0.672131
Recall:      0.788462
F1 Score:    0.725664
```



Our neural network performed poorly compared to some of our earlier ensemble methods, but let's see if doing cross validation will help us determine which model is truly the best.

Part 7: K-Fold Cross Validation

For logistic regression using PCA, our mean accuracy across folds is 75.53%. For ensemble method using bagging, our mean accuracy across folds is 74.50%. For neural net using MLP Classifier, our mean accuracy across folds is 71.28%. As we can see here, logistic regression using PCA was our highest-performing model.

Discussion

Overall, I wasn't too satisfied with the results I obtained. I think my mean accuracy across folds definitely could have been higher, and if given more time to experiment with different models and tweak the parameters, I think I could have improved my accuracy. Some other methods I would like to explore for this dataset are k-nearest neighbors, support vector machines, and AdaBoost (ensemble method).

I would recommend the UCLA hospital to use my best-performing prediction model, which was the logistic regression using PCA since logistic regression is generally a good classification model for binary classification problems like these. I would advise the UCLA hospital to take the results from the model with a grain of salt because the model is far from perfect, and it would be best to have a doctor or an expert in the field additionally review the patient's information history.

In terms of next steps for analytic work, I think that while this dataset was quite comprehensive, more information could help this model improve even more. For example, based on the CDC's list of potential risk factors, it may be helpful to know whether an individual has suffered from a stroke before, their blood pressure, their cholesterol levels, whether or not they have diabetes, whether or not they have sickle cell disease. There is also behavioral information that could be included in this dataset, such as diet, physical activity levels, and alcohol levels. Characteristics like race or ethnicity could also be a big indicator of whether someone will get a stroke.

Conclusion

This project dove into various models including logistic regression, principal component analysis, decision trees, random forests, bagging, and neural networks to try to come up with an effective prediction model of whether an individual will have a stroke. Each model was trained on its respective training dataset, and each model's performance was evaluated using the testing dataset. F1 score was used as the main performance metric as it takes both precision and recall into account. This problem was an imbalanced classification problem, so I downsampled the majority class. After running K-fold cross validation on the various models I explored in this project, the model that performed best was the logistic regression model with PCA, which had approximately a 76% accuracy.

Resources

- Stroke | cdc.gov
- https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb
- https://medium.com/analytics-vidhya/what-is-balance-and-imbalance-dataset-89e8d7f46bc5
- How to Handle Imbalanced Classes in Machine Learning (elitedatascience.com)
- Bagging and Random Forest for Imbalanced Classification (machinelearningmastery.com)
- Ensemble Learning, Bootstrap Aggregating (Bagging) and Boosting
- Scikit Learn Ensemble Learning, Bootstrap Aggregating (Bagging) and Boosting
- Bagging (Bootstrap Aggregation) - Overview, How It Works, Advantages (corporatefinanceinstitute.com)
- Chapter 10 Bagging | Hands-On Machine Learning with R (bradleyboehmke.github.io)
- sklearn.ensemble.BaggingClassifier — scikit-learn 0.24.2 documentation
- Multi Layer Perceptron | SKlearn | ipynb notebook example
- Activation Functions | Fundamentals Of Deep Learning (analyticsvidhya.com)